第3章 硬件描述语言Verilog HDL基础

3.1 学习要点

1. 硬件描述语言概述

硬件描述语言(Hardware Description Language,HDL)是一种用形式化方法来描述数字逻辑电路和系统的语言。利用这种语言,数字电路系统的设计可以从上层到下层、从抽象到具体逐层描述,用一系列分层次的模块实现复杂的数字系统。

硬件描述语言可以在结构级(也称逻辑门级)、行为级、寄存器传输级(Register Transfer Level,RTL)这几种不同的层次上对电路进行描述。通过逻辑综合,后两种层次的硬件描述语言源文件可以被转换到低抽象级别的门级描述。

硬件描述语言有两种用途:系统仿真和硬件实现。如果仅用于仿真,那么所有的语法和编程方法都可以使用。如果用于硬件实现,则必须保证程序"可综合"。也就是说,所有的硬件描述语言描述都可以用于仿真,但不是所有的硬件描述语言描述都能用于硬件实现。

2. Verilog 语言基本概念

1) Verilog 程序基本结构

以全加器为例,见图 3-1-1 和图 3-1-2,描述了一个名为 full_adder 的模块,模块(module)是 Verilog 的基本描述单位,用于描述逻辑电路的功能或结构,以及与子模块端口的关系。full_adder 模块有 5 个端口:3 个输入端口和 2 个输出端口。全加器为顶层模块,半加器为底层模块,通过模块例化调用底层模块。

```
//功能描述:全加器,由两个半加器组成
module  full_adder(
  input  Ai,   //输入端口
  input  Bi,   //输入端口
  input  Ci,   //低位进位,输入端口
  output Si,   //本位和,输出端口
  output Ciout);//向高位进位,输出端口
//线网说明
Wire    C0,A0,C1;
assign Ciout = C0|C1; //全加器输出
//第1个半加器例化
half_adder u0(
    .Ai(Ai),  //端口A
    .Bi(Bi),  //端口B
    .Si(A0),  //端口S
    .Ci(C0)); //端口C
//第2个半加器例化
half_adder u1(
    .Ai(A0),  //端口A
    .Bi(Ci),  //端口B
    .Si(Si),  //端口S
    .Ci(C1)); //端口C
endmodule
```

图 3-1-1　全加器 full_adder.v

```
//半加器
module half_adder(//模块名为 half_adder
  input  Ai, //输入端口
  input  Bi, //输入端口
  output Si, //输出端口和
  output Ci  //输出端口进位
  );
//用门电路实现半加器
  assign Si = Ai ^ Bi;//异或
  assign Ci = Ai & Bi;//与
endmodule    //模块结束关键字
```

图 3-1-2　半加器 half_adder.v

模块以关键词 module 开始,以关键词 endmodule 结尾,模块包括 4 部分:端口定义、I/O 说明、内部信号声明、功能定义。

2) Verilog 语言要素

在 Verilog 程序中,有各种符号,例如,标识符、操作符、字符串、注释等。

(1) 标识符。

标识符用于定义模块、端口、实例等名称。标识符可以是任意一组字母、数字、$ 符号和_(下画线)符号的组合,标识符的第一个字符必须是字母或者下画线,字符数不能多于 1024 个。标识符是区分字母大小写的。

(2) 关键词。

Verilog 语言内部已经使用的词称为关键词或保留字,关键词不能随便使用。需注意关键词都是小写字母。例如,always 是关键词,ALWAYS 不是关键词。

(3) 注释。

Verilog 程序的注释有 2 种:多行注释和单行注释。多行注释:以"/*"符号开始,到"*/"结束;在两个符号之间的语句都是注释语句。单行注释:以"//"开始到本行结束都属于注释语句,不允许续行。

(4) 逻辑值。

在 Verilog 中有 4 种逻辑值。

① 逻辑 0:表示低电平,在逻辑电路中通常接地;

② 逻辑 1:表示高电平,在逻辑电路中通常接电源;

③ 逻辑 x:表示逻辑状态不定,有可能是高电平,也有可能是低电平;

④ 逻辑 z:表示高阻态,是一个悬空状态。

(5) 常量。

在程序运行过程中,其值不能被改变的量称为常量。常量有整型常量、字符型常量、参数、线网(wire)和寄存器(reg)等数据类型。

3) Verilog 运算符

(1) 算术运算符。

算术运算符又称为二进制运算符,有下面 5 种。

① +:加法运算符,或正值运算符,如 a+b、+3;

② -:减法运算符,或负值运算符,如 a-3,-3;

③ *:乘法运算符,如 a*3;

④ /:除法运算符,如 5/3;

⑤ %:模运算符,也称为求余运算符,要求%两侧均为整型数据。

(2) 关系运算符。

在进行关系运算时,如果声明的关系是假的(false),则返回值是 0;如果声明的关系是真(true),则返回值是 1。关系运算符如下:

① a<b　　a 小于 b

② a>b　　a 大于 b

③ a<=b　　a 小于或等于 b

④ a>=b　　a 大于或等于 b

⑤ a==b　　a 逻辑相等 b

⑥ a!=b　　a 逻辑不等于 b

(3) 逻辑运算符。

逻辑运算符包括 3 种运算：&&(逻辑与)、||(逻辑或)、!(逻辑非)。

(4) 位逻辑运算符。

如果操作数长度不相等，那么长度短的操作数在高位补 0。按位逻辑运算有 5 种。

① ～：非运算；

② &：与运算，例如，4'b0100 & 4'b0110 运算结果为 4'b0100；

③ |：或运算，例如，4'b0100 | 4'b0110 运算结果为 4'b0110；

④ ^：异或运算，例如，4'b0100 ^ 4'b0110 运算结果为 4'b0010；

⑤ ～^，^～：同或运算，例如，4'b0100～^ 4'b0110 运算结果为 4'b1101。

(5) 条件运算符。

条件操作符是三目运算符，根据条件表达式的值选择表达式，形式如下：

条件 ? 表达式 1 : 表达式 2

如果条件为真(即值为 1)，则执行表达式 1；如果条件为假(值为 0)，则执行表达式 2。

(6) 连接运算符。

连接运算{}也称拼接运算，是将{}内的小表达式合并成大表达式，形式如下：

{expr1,expr2,…,exprN}

(7) 移位运算符。

在 Verilog 中，有两种移位运算符：<<(左移位运算符)和>>(右移位运算符)。执行移位运算时，操作数空出的位置填 0。

(8) 操作符优先级。

操作符优先级见表 3-1-1，顶部优先级最高，底部优先级最低；同一行的操作符具有相同的优先级。

表 3-1-1　操作符优先级

操　作　符	优　先　级
! ～	最高级
* / %	↓
+ −	↓
<< >>	↓
< <= > >=	↓
== != === !==	↓
&	↓
^ ^～	↓
|	↓
&&	↓
||	↓
? :	最低级

3. Verilog 行为语句

Verilog 行为语句有过程语句、块语句、赋值语句、条件语句、循环语句等，如表 3-1-2

所示。

表 3-1-2 Verilog 行为语句

类 别	语 句	可 综 合
过程语句	initial	
	always	是
块语句	串行块 begin end	是
	并行块 fork join	
赋值语句	持续赋值 assign	是
	过程赋值 = <=	是
条件语句	if else	是
	case	是
循环语句	for	范围必须是静态时可综合
	repeat	重复值是常数时可综合
	while	
	forever	
编译预处理	'define	是
	'include	是
	'ifdef 'else 'endif	是

1）赋值语句

在 Verilog 语言中，有持续赋值语句和过程赋值语句，过程赋值有非阻塞赋值和阻塞赋值。

（1）持续赋值语句。

assign 为持续赋值语句，主要对 wire 型变量赋值。

（2）过程赋值语句。

过程赋值语句主要对 reg 型变量赋值。

① **非阻塞赋值方式。**符号“<=”用于非阻塞赋值，非阻塞赋值语句在执行时不会阻塞后面的语句执行，也就是后面的赋值语句也同时执行。

② **阻塞赋值方式。**符号“=”用于阻塞的赋值，阻塞赋值“=”在 begin 和 end 之间的语句是顺序执行，属于串行语句。

2）条件语句

（1）if-else 语句。if 语句是用来判定所给定的条件是否满足，根据判定的结果决定执行给出的两种操作之一。

（2）case 语句。case 语句是一种多分支选择语句，if 语句只有两个分支可供选择，而实际问题中常常需要用到多分支选择，case 语句直接处理多分支选择。

3）循环语句

在 Verilog 中有 4 种类型的循环语句，用来控制执行语句的执行次数。

（1）forever：连续地执行语句，用在 initial 块中，生成时钟等周期性波形。

（2）repeat：反复执行一条语句 n 次。

（3）while：执行一条语句直到某个条件不满足。如果一开始条件就不满足（为假），

则语句一次也不能被执行。

(4) for：有条件循环语句。

4. Verilog 语言的描述语句

一个复杂的数字逻辑系统的 Verilog 模型是由若干个 Verilog 模块构成的，每个模块又由若干个子模块构成。用 Verilog 描述的数字逻辑电路就是该逻辑电路的 Verilog 模型(也称模块)，是 Verilog 的基本描述单位。

模块的几种描述形式(也称建模方式)包括行为级描述形式、结构级描述形式以及数据流描述形式。如果从逻辑电路结构描述电路模块，则称为结构描述形式；如果对线性变量进行操作，则称为数据流描述形式；如果从功能和行为描述电路模块，则称为行为级描述形式。

1) 数据流描述形式

数据流描述是用连续赋值语句 assign 实现，主要实现组合逻辑电路。

2) 结构描述形式

结构描述就是在设计中实例化已有的功能模块，这些模块包括 Verilog 语言自带的和用户自行开发的。

3) 行为描述形式

行为描述是对实体的数学模型的描述，只需要描述清楚输入和输出信号的行为，其抽象程度高于结构描述。可综合的行为描述大多采用 always 过程语句，适用于时序逻辑电路，也适用于组合逻辑电路。

5. Verilog 仿真验证

仿真(Simulation)是对所设计逻辑电路的一种检测方法，Verilog HDL 不仅提供了设计描述的能力，而且提供了对激励、响应和设计验证的建模能力。

要进行逻辑电路的仿真就需要用仿真器，ModelSim 是常用的仿真器。在 Verilog 语言中，将测试程序称为 testbench，意思是测试平台。测试程序与逻辑电路描述的 Verilog 程序类似，也是由模块组成，模块将描述测试激励、被测试模块和测试结果。测试激励用初始化语句产生，被测试模块在测试程序中作为一个实例嵌入，通过被测试模块的输出响应，判断模块功能是否正确。

3.2 例题解析

例 3-1 画出如图 3-2-1 所示的 Verilog 语句描述的逻辑电路，其中 A、B、C、D、E 是输入，F 是输出。

解： 根据 Verilog 语句描述，可以画出如图 3-2-2 所示的逻辑电路图。

例 3-2 图 3-2-3 是逻辑电路的 Verilog 源程序，图 3-2-4 是逻辑电路的 Verilog 仿真源程序，确定仿真结果 F 的值。

```
module L3_1 (A, B, C, D, E, F);
//端口定义
input  wire A, B, C, D, E;
output wire F;
//线网定义
wire G,H,I;
//逻辑功能定义
assign G  =  A + B;
assign H  =  ~C & D;
assign I  =  G + H;
assign F  =  I & E;
endmodule
```

图 3-2-1　例 3-1 Verilog 源程序

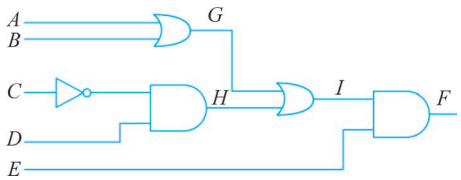

图 3-2-2　例 3-1 逻辑电路

```
module L3_2 (
input  wire [7:0] A,
input  wire [4:0] B,
output wire [8:0] F
);
//逻辑功能定义
assign F = {~B,4'b0111} & {A,1'b1}
           | {1'b1,A};
endmodule
```

图 3-2-3　例 3-2 Verilog 源程序

```
`timescale 1 ns/ 100 ps
module L3_2_tb();
reg  [7:0] A;
reg  [4:0] B;
wire [8:0] F;
L3_2 DUT (
    .A(A),
    .B(B),
    .F(F) );
initial begin
#100 $stop;
end
initial begin
    A = 8'b00101101;
    B = 5'b10010;
end
endmodule
```

图 3-2-4　例 3-2 Verilog 仿真源程序

解：在如图 3-2-3 所示的 Verilog 源程序中，语句“assign F = {～B,4'b0111} & {A,1'b1}| {1'b1,A};”中的并置运算有{～B,4'b0111}、{A,1'b1}、{1'b1,A}。

在如图 3-2-4 所示的 Verilog 仿真源程序中，语 句“A = 8'b00101101,B = 5'b10010”中的并置运算结果有{～B,4'b0111}为“011010111”、{A,1'b1}为“001011011”、{1'b1,A}为“100101101”。

F =“011010111” AND “001011011” OR “100101101” = 101111111

例 3-3　逻辑电路如图 3-2-5 所示，编写实现逻辑电路的 Verilog 源程序，通过 ModelSim 确定该逻辑电路的逻辑功能。

解：由如图 3-2-5 所示的逻辑电路，编写实现逻辑电路的 Verilog 源程序，如图 3-2-6 所示。Verilog 仿真源程序见图 3-2-7，ModelSim 仿真结果见图 3-2-8。

由图 3-2-8 可知，当 $D=0$，$E=0$ 时，输出 $F=A$；当 $D=1$，$E=0$ 时，输出 $F=B$；当 $E=1$ 时，输出 $F=C$。该逻辑电路的功能是数据选择器，输入数据为 A、B、C，输入控制端为 D、E。

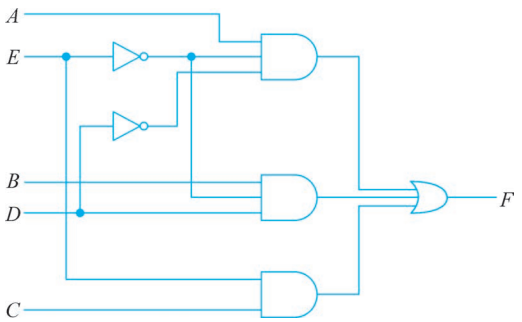

图 3-2-5　例 3-3 逻辑电路

```
module L3_3 (A, B, C, D, E, F);
//端口定义
input  wire A, B, C, D, E;
output wire F;
//线网定义
wire G,H,I;
//逻辑功能定义
assign G  = A & ~E & ~D;
assign H  = B & ~E &  D;
assign I  = C &  E;
assign F  = G | H | I;
endmodule
```

图 3-2-6　例 3-3 Verilog 源程序

```
`timescale 1ns/100ps
module L3_3_tb();
reg A;
reg B;
reg C;
reg D;
reg E;
wire F;
L3_3 DUT (
    .A(A),
    .B(B),
    .C(C),
    .D(D),
    .E(E),
    .F(F));
initial
begin
#200 $stop;
end
always begin
    A = 1'b0;
    A = #20 1'b1;
    #20;
end
```

```
always begin
    B = 1'b0;
    B = #10 1'b1;
    #10;
end
always begin
    C = 1'b0;
    C = #5 1'b1;
    #5;
end
initial begin
    D = 1'b0;
    D = #40 1'b1;
    D = #40 1'b0;
    D = #40 1'b1;
    D = #40 1'b0;
end
initial begin
    E = 1'b0;
    E = #80 1'b1;
    E = #80 1'b0;
end
endmodule
```

图 3-2-7　例 3-3 Verilog 仿真源程序

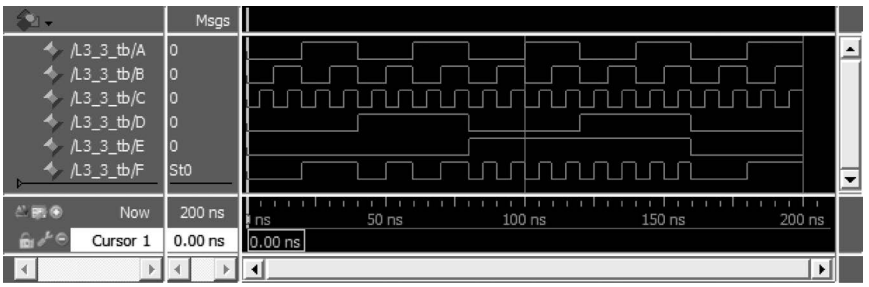

图 3-2-8　例 3-3 ModelSim 仿真结果

3.3 习题解答

3-1 用 Verilog 语言描述如图 3-3-1 所示的逻辑电路。

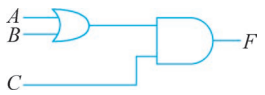

图 3-3-1 习题 3-1 逻辑电路

解： 由如图 3-3-1 所示的逻辑电路，可以写出 Verilog 语言的源程序，如图 3-3-2 所示。

3-2 用 Verilog 实现二输入异或门 $F=A\oplus B$ 的逻辑功能。

解： 由表达式 $F=A\oplus B$，可以写出 Verilog 语言的源程序，如图 3-3-3 所示。

```
module T3_1 (A, B, C, F);
//端口定义
    input  A, B, C;
    output F;
//逻辑功能定义
    assign F  = (A|B)&C;
endmodule
```

图 3-3-2 习题 3-1 Verilog 源程序

```
module T3_2 (A, B, F);
//端口定义
    input  A, B;
    output F;
//逻辑功能定义
    assign F  = A^B;
endmodule
```

图 3-3-3 习题 3-2 Verilog 源程序

3-3 用 Verilog 语言实现如图 3-3-4 所示的逻辑电路。编写如图 3-3-5 所示输入波形的 ModelSim 仿真 Verilog 程序，给出 Q 和 Q_n 的仿真结果。

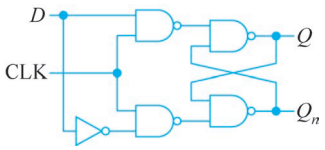

图 3-3-4 习题 3-3 逻辑电路

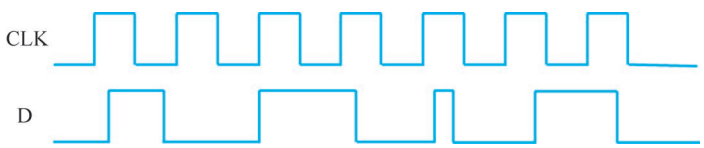

图 3-3-5 习题 3-3 输入波形

解： 由如图 3-3-4 所示的逻辑电路，可以写出 Verilog 语言的源程序如图 3-3-6 所示。由如图 3-3-5 所示的输入波形，可以写出 Verilog 语言的仿真源程序如图 3-3-7 所示。ModelSim 仿真结果见图 3-3-8，观察仿真结果，说明门控 D 锁存器的输出 Q 和 Q_n 是正确的。

3-4 在 Verilog 源程序文件中，怎样标明注释？

解： Verilog 程序的注释有两种：多行注释和单行注释。

（1）多行注释：以“/*”符号开始，到“*/”结束；在两个符号之间的语句都是注释语句。

（2）单行注释：以“//”开始到本行结束都属于注释语句，不允许续行。

3-5 用 Verilog 语言实现如图 3-3-9 所示的 4 选 1 数据选择器。

解： 由如图 3-3-9 所示的逻辑电路，可以写出 Verilog 语言的源程序如图 3-3-10 所示。

```
module T3_3(
input  wire CLK,
input  wire D,
output wire Q,
output wire Qn);
//功能定义
assign Q  = ~(Qn & ~(CLK & D));
assign Qn = ~(Q  & ~(CLK &~D));
endmodule
```

图 3-3-6　习题 3-3 Verilog 源程序

```
`timescale 1 ns/ 100 ps
module T3_3_tb();
reg CLK;
reg D;
wire Q;
wire Qn;
T3_3 DUT (
    .CLK(CLK),
    .D(D),
    .Q(Q),
    .Qn(Qn));
initial begin
CLK = 1'b0;
#750 $stop;
end
always #50 CLK = ~ CLK;
initial begin
    D = 1'b0;
    D = #70  1'b1;
    D = #70  1'b0;
    D = #110 1'b1;
    D = #120 1'b0;
    D = #90  1'b1;
    D = #30  1'b0;
    D = #90  1'b1;
    D = #110 1'b0;
end
endmodule
```

图 3-3-7　习题 3-3 Verilog 仿真源程序

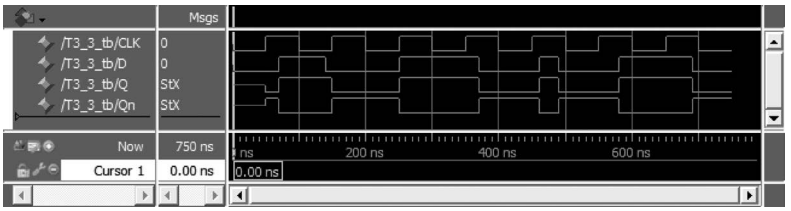

图 3-3-8　习题 3-3 ModelSim 仿真结果

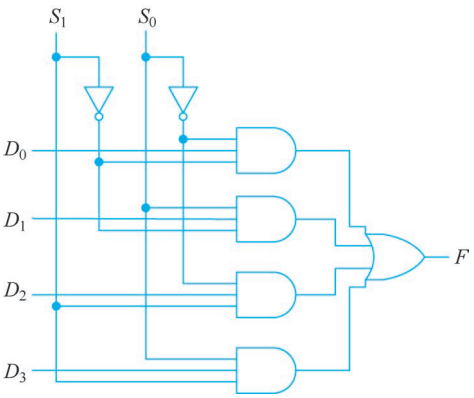

图 3-3-9

```
//由门电路组成的4选1数据选择器
module T3_5(
    input  wire  S0, //输入选择低位
    input  wire  S1, //输入选择高位
    input  wire  D0, //输入低位
    input  wire  D1,
    input  wire  D2,
    input  wire  D3,  //输入高位
    output wire  F    //输出
    );
//线网类型
wire    FD0,FD1,FD2,FD3;
//功能定义
assign  FD0 =  ~S1 & ~S0 & D0;
assign  FD1 =  ~S1 &  S0 & D1;
assign  FD2 =   S1 & ~S0 & D2;
assign  FD3 =   S1 &  S0 & D3;
assign  F = FD0 | FD1 | FD2 | FD3;
endmodule
```

图 3-3-10　习题 3-5 Verilog 源程序